Introduction to Computer Organization – Fall 2023

Lab 5

Due: @11:55 PM, Wed October 4th

The following assignment is intended to be completed during your assigned lab period. One member of your group must submit the assignment to Gradescope by the posted deadline and indicate your group members when submitting the assignment. **Each group member must be present during the scheduled lab period in order to receive credit.**

Group names and uniqnames

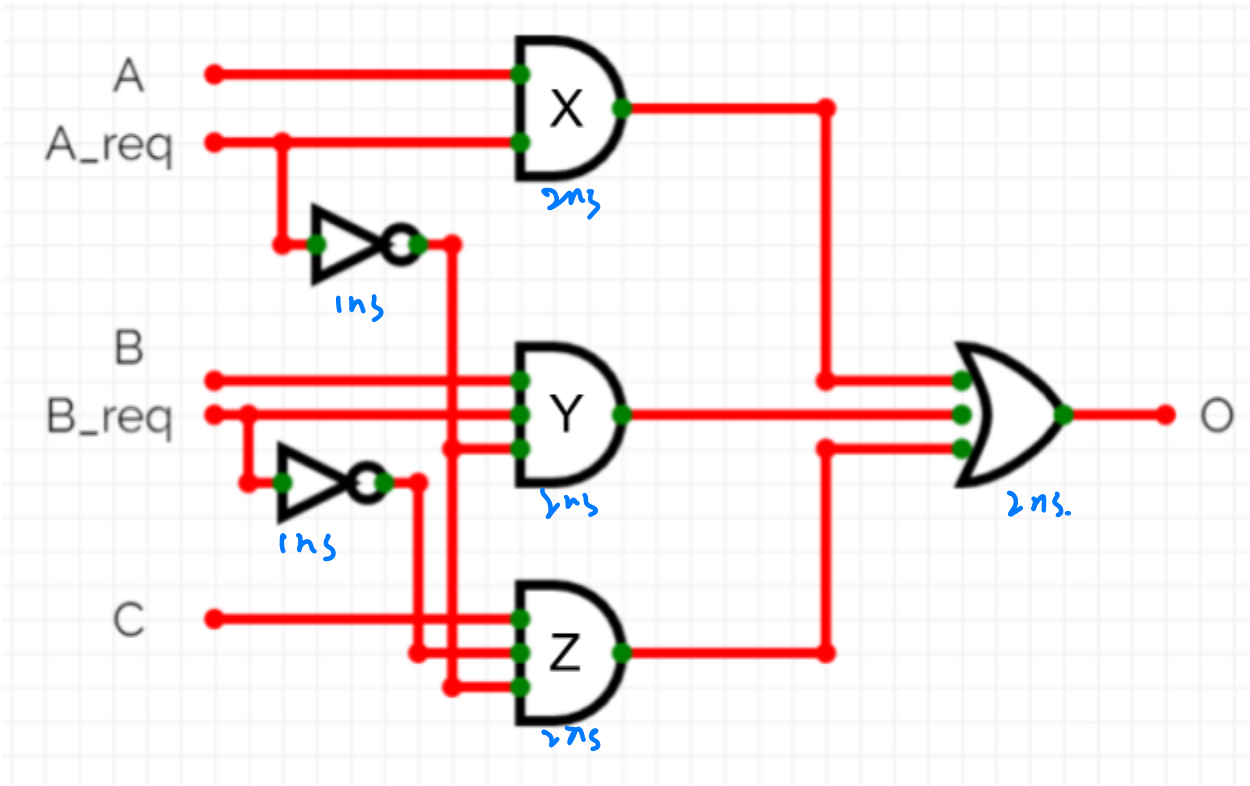
Pod Member Name	**Uniqname**
Yuzhen chen	yuzhech
Therron Montgomery	therronm
Yuhan Zhang	zyuhan

For each of the following problems, one person should act as the "scribe" and log the discussions of the group. You should rotate who is the scribe for each problem and indicate in the given space.

Problem 1: Circuit Delay *[15 Points]* Topics: Timing Diagrams, Propagation Delay

Scribe: **[Scribe's name here]**

Consider the following naive implementation of a priority selector. This priority selector takes 3 data inputs, “A,” “B,” and “C” and one data output, “O.” It has 2 requester inputs, “A_req,” “B_req”. If A_req is asserted, O outputs the value of A. If A_req is not asserted but B_req is, O outputs the value of B. If no requesters are asserted, C is outputted.



Assume that NOT gates have a 1ns delay, and that OR and AND gates have a 2ns delay regardless of the number of input wires. Assume no wire delay.

Say that for an infinite amount of time before we start measuring, all input signals are “1.”

- At 1ns, B_req goes to “0”
- At 2ns, A_req goes to “0”
- At 6ns, C goes to “0”.
- At 8ns, A goes to “0”.
- At 9ns, A_req and B_req go to “1”.

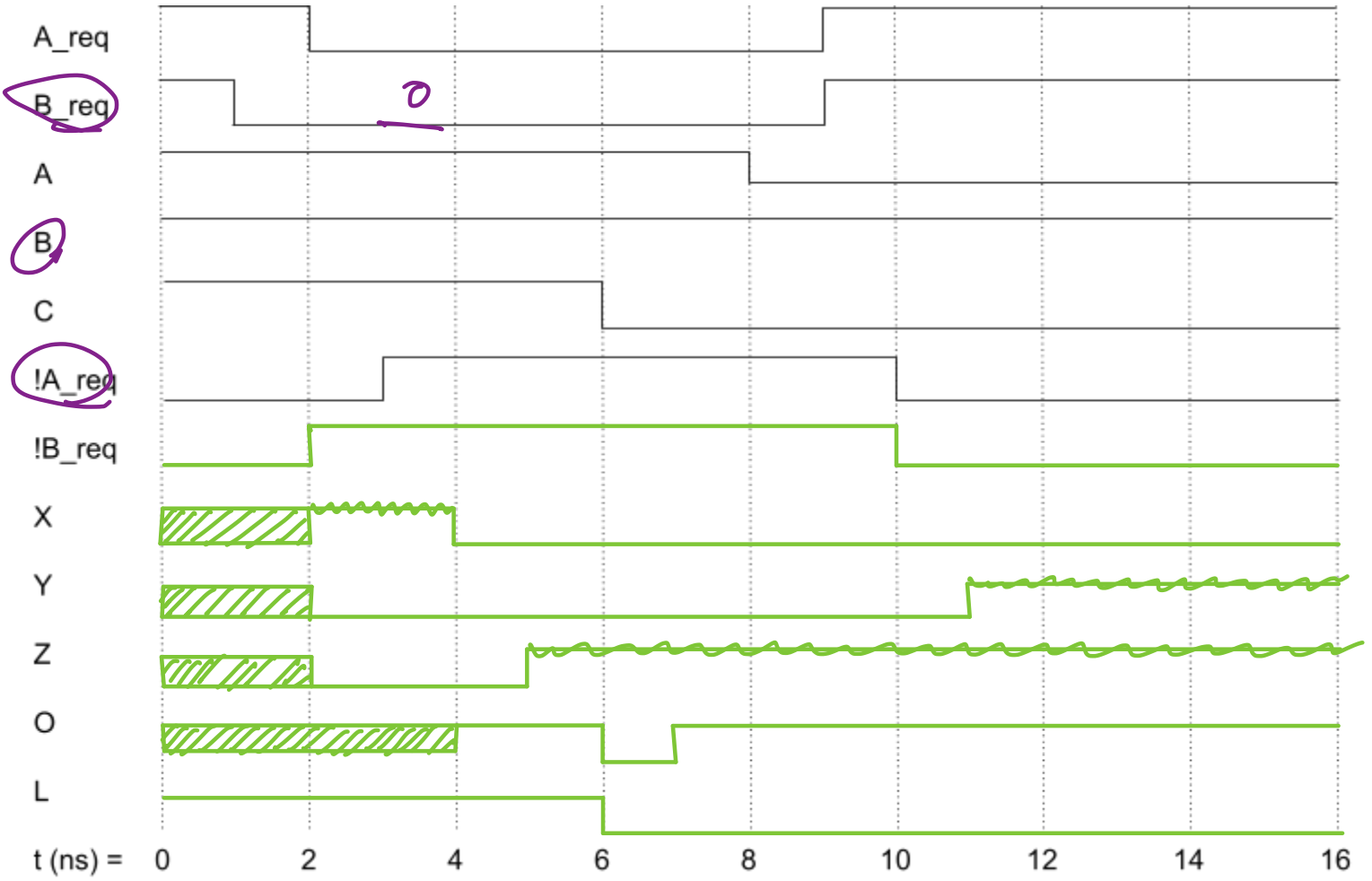
X = A · A_req

Y = B · B_req · A_req'

Z = C · B_req' · A_req'

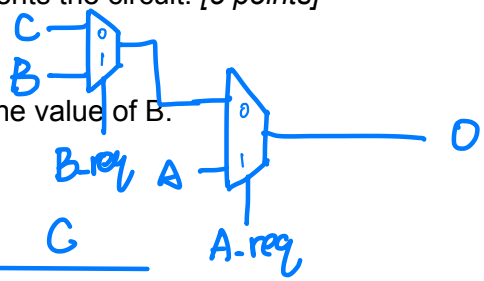
O = X + Y + Z

a. Draw a timing diagram from t=0ns to t=16ns showing the input signals (A, B, C), requester signals (A_req, B_req), the output signal (O), and all intermediate gates (!A_req, !B_req, X, Y, Z). The inputs, requesters, and 1 intermediate gate have been done for you. Also include a line "L" (for Logic) depicting what the output signal would be with no delay. *[10 points]*
For drawings, you may draw by hand and insert a photo of your work, or merge PDFs before submission, or just digitally modify the drawing below.



b. Draw a circuit using only two 2-input Muxes that implements the circuit. *[5 points]*
The definition of the 3-input priority selector is repeated:

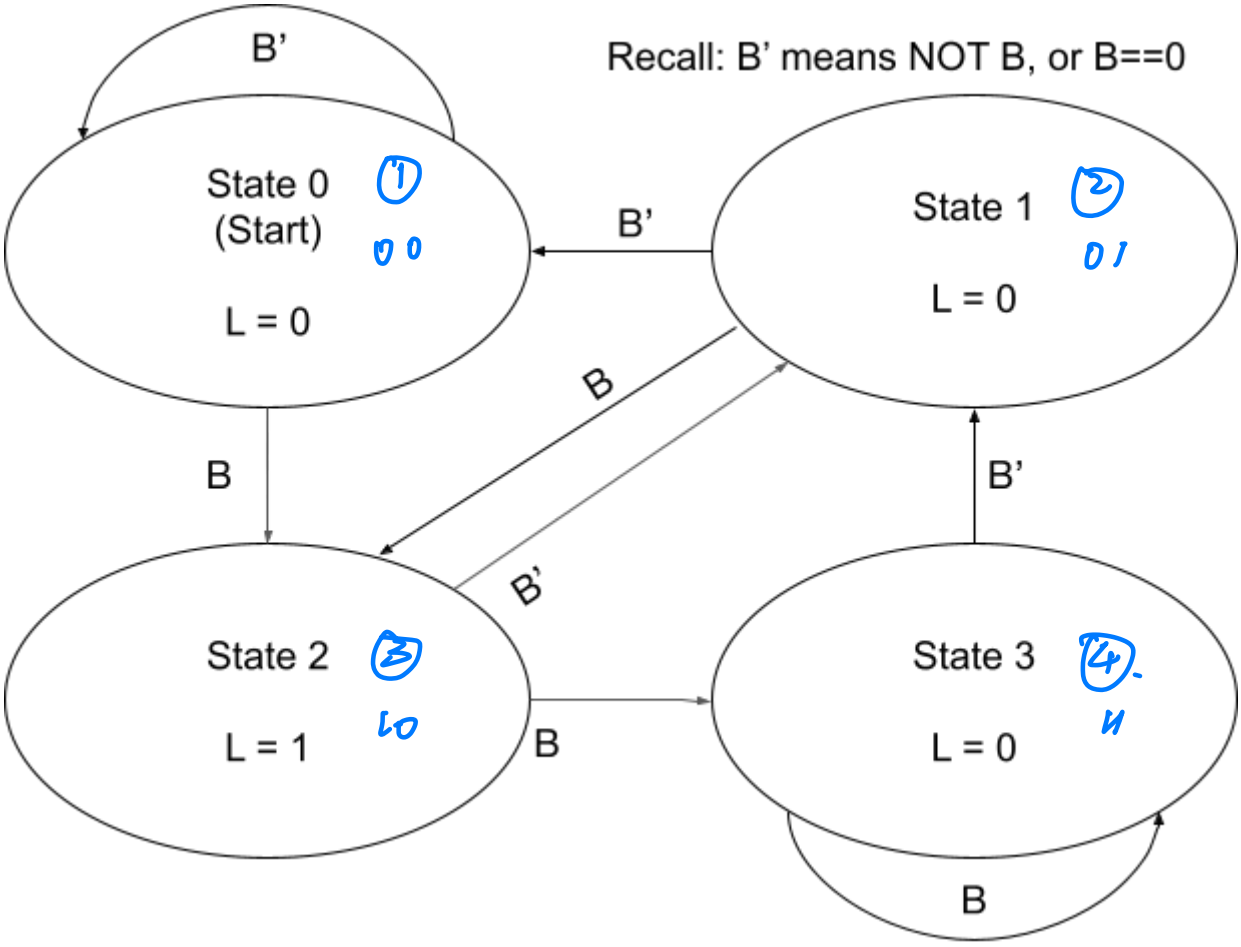
- If A_req is asserted, O outputs the value of A.
- If A_req is not asserted but B_req is, O outputs the value of B.
- If no requesters are asserted, C is outputted.



A_req	B_req	A	B	C
0	0	0	0	1
0	1	0	1	0
1	0	1	0	0
1	1	1	0	0

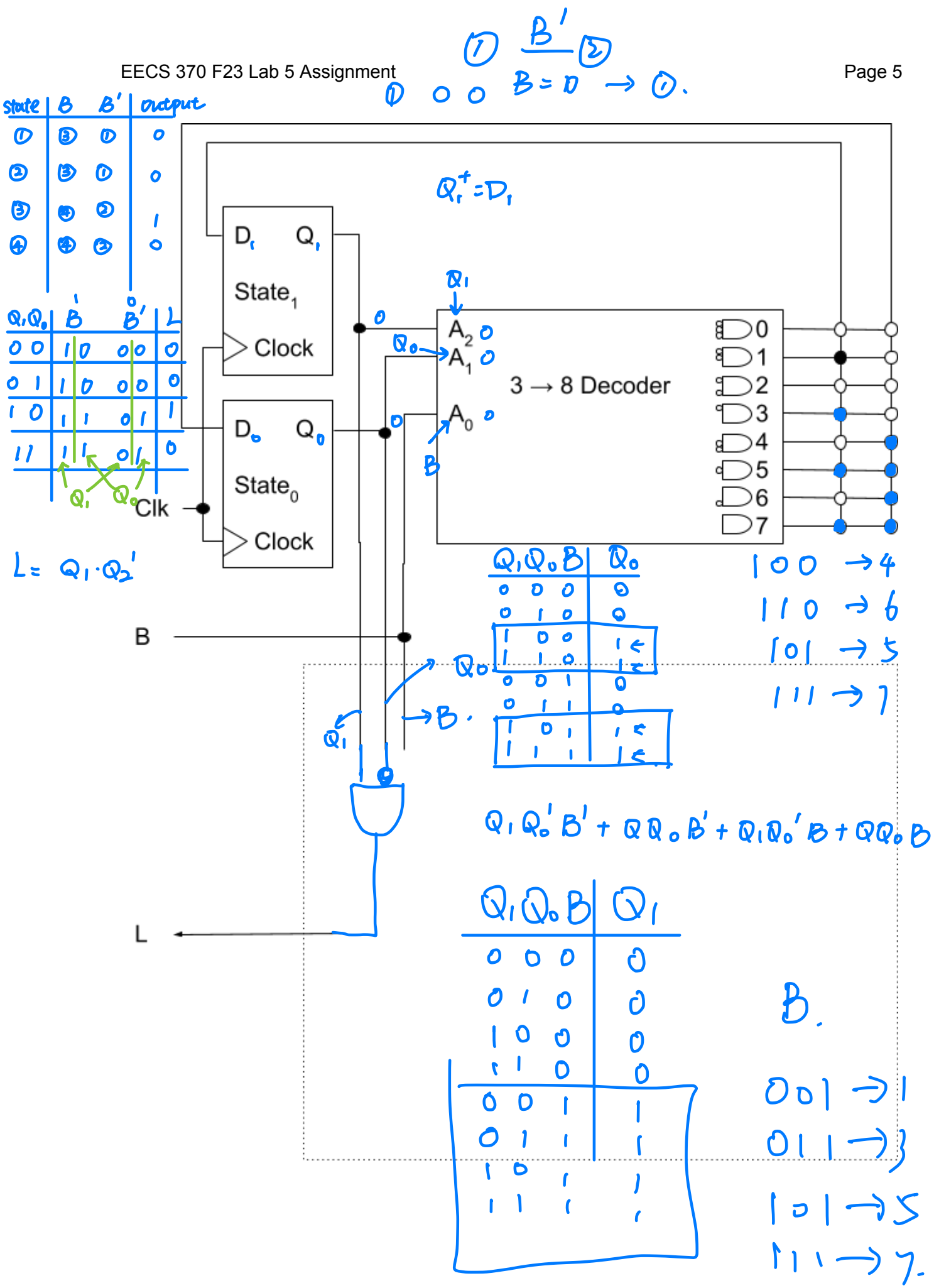
Problem 2: Finely Stated *[15 Points]* Topics: FSM, Flip Flops, Decoders
Scribe: **[Scribe's name here]**

Consider the finite state machine diagram below. This FSM tracks the last 2 values of the input B, sets an output L to 1 if a code is received.



On the next page, convert the Finite State Machine into a circuit. Fill in the provided bubbles in the control ROM to determine the next state *[7.5 points]*. Inside the dotted box, add wires and logic gates to complete the rest of the logic for the output L *[7.5 points]*.

- Assume the clock is connected but not shown.
- Assume the flip flops have a starting value of 0.
- Do not add outputs to the control ROM, and do not modify anything outside the ROM bubbles and the dotted box.
- The decoder inputs are {$State_1$, $State_0$, B}
- Filling in one of the control ROM bubbles means there is a connection, so current flows and we get a 1. Otherwise, the wires are disconnected and we get a 0.
    - For example, state 0 is done for you in the first 2 rows. When the FSM is in state 0 and B=0, the next state is 0, so both Next State bits are 0 and not filled in. But when the FSM is in state 0 and B=1, the next state is 2 (binary 10), so the bubble for the Next $State_1$ bit is filled in while the bubble for Next $State_0$ is not.


D1 Q1
State1
Clock
D0 Q0
State0
Clock
Clk
A2
A1
A0
3 → 8 Decoder
0
1
2
3
4
5
6
7
B
L

Problem 3: LC2K ABI *[20 Points]* Topics (Review from Lab 4): ABI, Functions, Loops
Scribe: **[Scribe's name here]**

Convert this function to LC2K using the following ABI:

- r0: always 0
- r1 - r2: function arguments, caller-save
- r3 - r4, r6: caller-save
- r5: stack pointer
- r7: return address (caller-save)

Assume the stack grows "up". If you want to store a new variable on the stack, you should write it to an address offset by r5, and then increment r5 by the appropriate amount. Make sure you decrement it back to the original value before returning from the function so that the calling function can access its own local variables.

Please submit typed LC2K code (which won't be autograded) - no handwritten code.

```
void start() {
     int a = -1, b = 2, c = 3;
     for (int i = 3; i != 0; --i) {
          mystery(i, a);
          c = i + b;
     }
}
```

reg 2 = -1 ← a.
reg 3 = 1 ← increasement
✓reg 4 = 2 ← b
reg 6 = 3 ← c.
✓reg 1 = 3 ← i

```
Start lw    0    2    neg1  //a
      lw    0    3    one   //inc
      add   3    3    4     //b
      add   3    4    6     //c
      add   0    6    1     //i
for   beq   3    0    done  //check.
      sw    5    7    stack // store return address of caller
      add   3    5    5     // sp ++
      sw    5    1    stack // store reg 1 (i) in to stack
      add   3    5    5     // sp++.
      sw    5    4    stack // store reg 4 (b) into stack.
      add   3    5    5     // sp++.
      lw    0    3    fcall
      jalr  3    7
done  halt
fcall .fill mystery.
one   .fill 1
neg1  .fill -1
```

reg1 → a = 1 reg6 = one 解决 update Stack 的最新位置.

```
void start() {
    int a = 1;
    a += mystery(a);
}
```

保 reg1 reg7

```
start  lw  0 1 one
       lw  0 6 one
       sw  5 7 stack
       add 5 6 5
       sw  5 1 stack
       add 5 6 5
       lw  0 6 fcall
       jar 6 7
       lw  0 6 negal
       add 5 6 5
       lw  5 1 stack
       add 5 6 5
       lw  5 7 stack
       jalr  7 6
```

用 reg5 来记录目前位置和 stack 相差多少.

reg5 ↑

Stack → reg7

```
one  .fill 1
fcall .fill Mystery
negal .fill -1
```